



Intel® Inspector XE 2013

Memory Checker
Thread Checker
Static Analysis
Pointer Checker

Deliver More Reliable Applications

Intel® Inspector XE and Intel® Parallel Studio XE family of suites

Dynamic Analysis

Memory Errors

Problems		
ID ▲	Problem	So
P1	Mismatched allocation...	fin
P2	Invalid memory access	fin
P3	Memory leak	fin

- Invalid Accesses
- Memory Leaks
- Uninit. Memory Accesses

Threading Errors

Timeline	
main (20940) (10940)	
thread_video (4492) (4492)	
Write: winvideo.h:270	

- Races
- Deadlocks
- Cross Stack References

- Multiple tools
- One common user interface
- Easy workflow for developers
- Windows & Linux

Static Analysis

Code & Security Errors

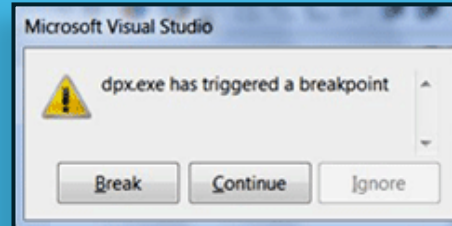
Code Locations: Divide by zero (possible)		
Description	Source	Function
Divide by zero	cylinder.cpp:131	void cylinder_
129	VCross(&rc, &cyl->axis,	
130	VDOT(t, 0, n);	
131	t = - t / ln;	

- Buffer over/under flows
- Incorrect pointer usage
- Over 250 error types...

Pointer Checker

Pointer Errors

NEW



- Out of bounds accesses
- Dangling pointers

**Find errors earlier
with less effort**

Static Analysis & Pointer Checker are only available in the Parallel Studio XE family of suites. Not sold separately.

Dynamic Analysis Finds Memory & Threading Errors

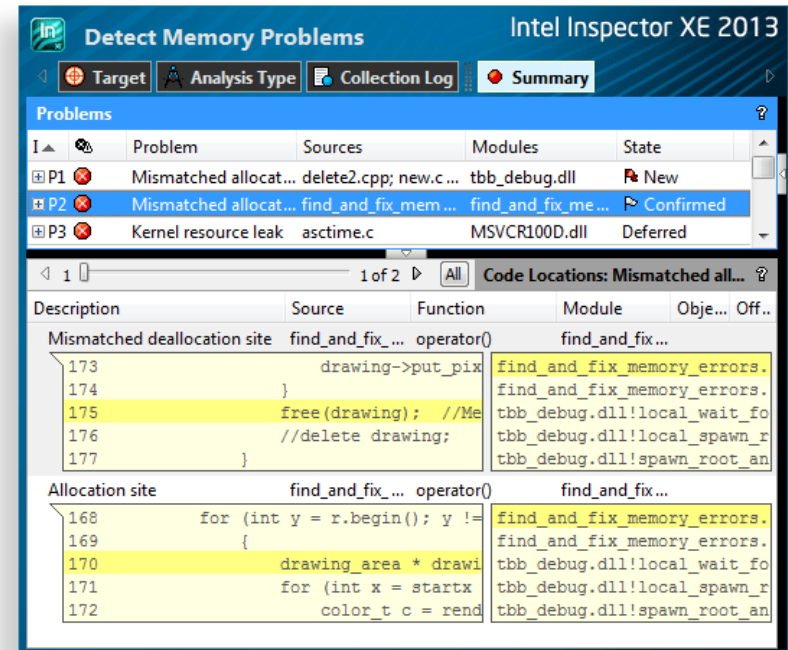
Intel® Inspector XE 2013

Find and eliminate errors

- Memory leaks, invalid access...
- Races & deadlocks
- C, C++, C#, F# and Fortran (or any mix)

Simple, Reliable, Accurate

- No special recompiles
Use any build, any compiler
- Analyzes dynamically generated or linked code
- Inspects third party libraries where source is unavailable
- Productive user interface
- Command line for automated regression analysis



Clicking an error instantly displays source code snippets and the call stack


Easy to fit into your existing process


New for 2013!

Intel® Inspector XE 2013 Dynamic Memory & Thread Analysis

Heap Growth Analysis

Heap Growth Analysis

 Set Transaction Start

 Set Transaction End

Diagnose heap growth. Get a list of memory allocations not freed in an interval set with the GUI or an API.

Improved Error Suppression

Stack frame match
☒ Best stack frame ☐ Top stack frame ☐ Any stack frame

Code location(s) that comprise the rule:

Problem	Code Location Description	Module/Function/Source/Line
<input checked="" type="checkbox"/> Memory leak	Allocation site	find_and_fix_memory_errors.exe/operator() - f...

General

☐ Any problem
☐ Any description







Stack frame

find_and_fix_memory_errors.exe/operator() - find_and_fix_memory_errors.cpp:163

☐ Any module ☐ Any function ☐ Any source ☐ Any line

More precise and team shareable. Choose which stack frame to suppress. Eliminate the false, not the real errors.

Debugger Breakpoints

Problems			
ID		Problem	Sources
P1		Mismatched allocation	
P2		Invalid memory access	<div>View Source Edit Source Copy to Clipboard Explain Problem Create Problem Report... Debug This Problem</div>
P3		Memory leak	
P4		Memory leak	
P5		Memory leak	
P6		Memory growth	

Diagnose the problem. Break into the debugger just before the error occurs. Examine the variables and threads.

Pause/Resume Collection


```
__itt_suppress_push(__itt_suppress_threading_errors);  
/* Any threading errors here are ignored */  
__itt_suppress_pop();  
/* Any threading errors here are seen */
```

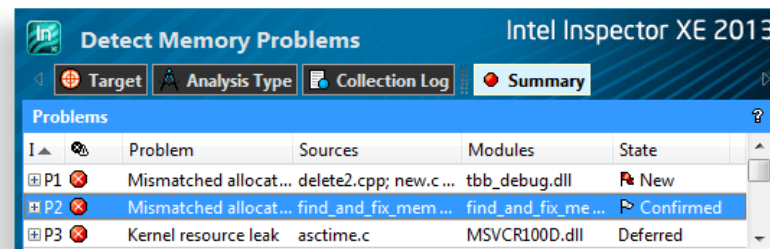
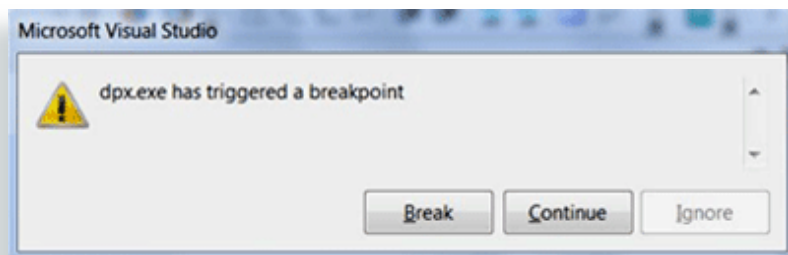
Speed-up analysis by limiting its scope. Turn on analysis only during the execution of the suspected problem.

Find and diagnose errors with less effort.

Pointer Checker and Memory Checker

Intel Parallel Studio XE family of suites

Pointer Checker 	Memory Checker
Recompile with Intel® Compiler	Use any build, any compiler
Lower overhead	Higher overhead
Only finds pointer errors	Finds multiple error types
One error at a time	GUI sorts multiple errors
Traceback: Source file + Line #	Traceback: Shows source code
Triggers debugger breakpoint	Triggers debugger breakpoint



Two great ways to create more reliable software

Static Analysis Finds Coding and Security Errors

Intel® Parallel Studio XE 2013 Family of Suites

Find over 250 error types

- Incorrect directives, memory leaks, pointer and array errors, buffer overflows, uninitialized variables...

Easier to use

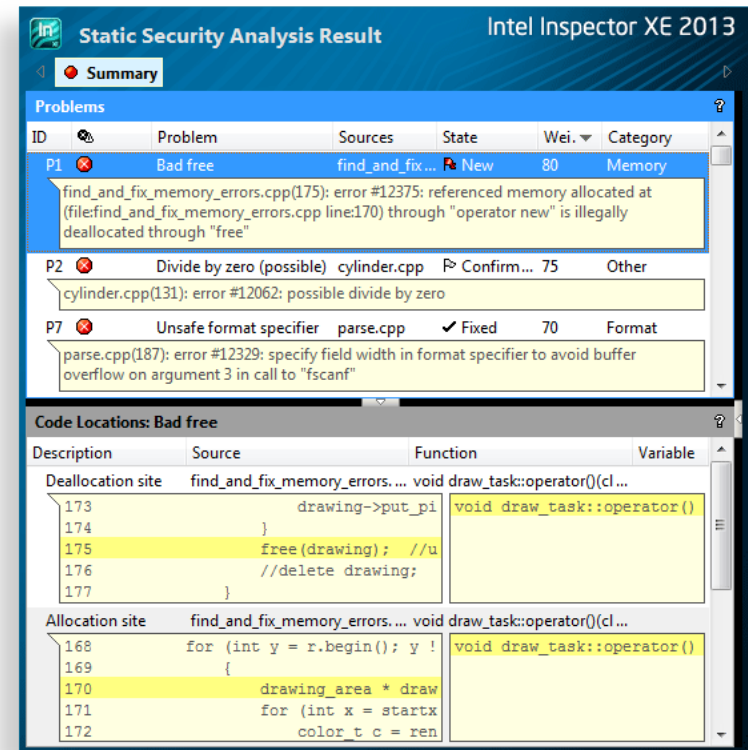
- Choose your priority:
 - Minimize false errors
 - Maximize error detection
- Hierarchical navigation of results
- Share comments with the team

Increased Accuracy & Speed

- Detect errors without all source files
- Better scaling with large code bases

Code Complexity Metrics

- Find code likely to be less reliable



Clicking an error instantly displays source code snippets and traceback. Available for C, C++ and Fortran.

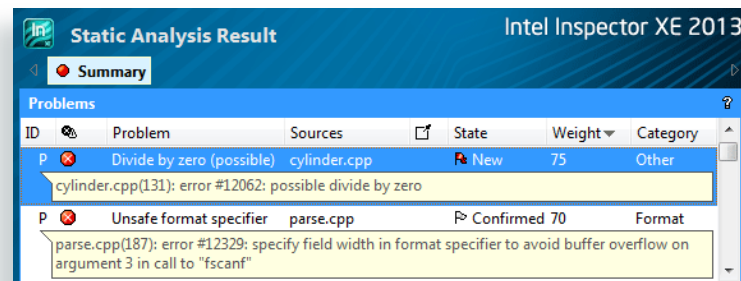
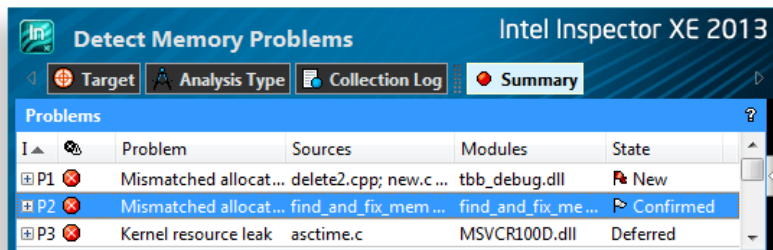
Find Errors and Harden your Security

Static Analysis is only available in the Parallel Studio XE family of suites. It is not sold separately.

Dynamic Analysis Complements Static Analysis

In Intel® Parallel Studio XE family suites

Dynamic Analysis	Static Analysis
Use any build, any compiler	Rebuild with Intel® Compiler (Keep your existing compiler for code generation.)
Fewer false errors. Only active code paths are analyzed.	Comprehensive, but more false errors. Not limited by test cases.
Analyze 3 rd party code	n/a – Source required
Can trigger debugger breakpoint	n/a – No diagnostic capability
Slow (1x – 20x - 100x workload)	Fast (no workload, “slow” build)
Memory & Threading Errors	Memory, Code & Security Errors



Two great ways to create more reliable software

User Interface

Intel® Inspector XE

Locate Memory Problems Intel Inspector XE 2013

Target Analysis Type Collection Log Summary

Problem	Sources	Object Size	State
Mismatched allocat...	find_and_fix_memory_erro ...		Confirmed
Invalid memory acc...	find_and_fix_memory_erro ...		Not fixed
Memory not deallo...	api.cpp; util.cpp; video.cpp		Confirmed
Memory leak	find_and_fix_memory_erro ...	1344	Deferred
Memory leak	find_and_fix_memory_erro ...	784	Fixed
Memory leak	find_and_fix_memory_erro ...	672	New
Memory leak	find_and_fix_memory_erro ...	1120	New

Filters Sort

Source

Source	Count
api.cpp	1 item(s)
find_and_fix_memory_errors.cpp	6 item(s)
util.cpp	1 item(s)
video.cpp	1 item(s)

State

State	Count
Confirmed	2 item(s)
Deferred	1 item(s)
Fixed	1 item(s)
New	2 item(s)

Code snippets displayed for selected problem

Description Source Funct... Module Object ... Offset

Mismatched deal... find_and_fix_memo ... opera ... find_and_fix_memo ...

```
173 drawing->put_p find_and_fix_memory_errors
174 } find_and_fix_memory_errors
175 free(drawing); // tbb_debug.dll!local_wait_f
176 //delete drawing; tbb_debug.dll!process - ar
177 } tbb_debug.dll!process - ma
```

Timeline shows when error occurred

threadstartex (9340) (9340)

Problem States:

New, Not Fixed, Fixed, Confirmed, Not a problem, Regression

Filters let you focus on a module, or error type, or..

Double Click for Source & Call Stack

Intel® Inspector XE

Call Stack

Source code locations displayed for selected problem

The screenshot displays the Intel Inspector XE 2013 interface. The top bar shows the title 'Mismatched allocation/deallocation' and the version 'Intel Inspector XE 2013'. Below the bar are tabs for 'Target', 'Analysis Type', 'Collection Log', 'Summary', and 'Sources'. The main window is divided into two panes. The left pane shows the source code of 'find_and_fix_memory_errors.cpp' with a blue highlight on line 170: `drawing_area * drawing = new drawing_area(startx, total`. The right pane shows the call stack for the selected line, listing several frames including 'find_and_fix_memory_errors.exe!operator()', 'find_and_fix_memory_errors.exe!execute', and 'tbb_debug.dll!local_wait_for_all - custom'. A yellow arrow points from the 'Call Stack' label above to the right pane. Another yellow arrow points from the 'Source code locations displayed for selected problem' label to the left pane. A third yellow arrow points from the 'Call Stack' label to the right pane. A fourth yellow arrow points from the 'Source code locations displayed for selected problem' label to the left pane.

Mismatched allocation/deallocation

Intel Inspector XE 2013

Target Analysis Type Collection Log Summary Sources

Mismatched deallocation site - Thread threadstartex (9340) (find_and_fix_memory_errors.exe!operator() - find_and_fix_memory_errors...

find_and_fix_memory_errors.cpp Disassembly (find_and_fix_memory_errors.exe!0x26e6)

170 drawing_area * drawing = new drawing_area(startx, total

171 for (int x = startx ; x < stopx; x++) {

172 color_t c = render_one_pixel (x, y, local_mbox, ser

173 drawing->put_pixel(c);

174 }

175 free(drawing); //Memory Error: use delete instead of

176 //delete drawing;

177 }

178 if(!video->next_frame()) return;

179 }

Allocation site - Thread threadstartex (9340) (find_and_fix_memory_errors.exe!operator() - find_and_fix_memory_errors.cpp:170)

find_and_fix_memory_errors.cpp Disassembly (find_and_fix_memory_errors.exe!0x2623)

167 for (int y = r.begin(); y != r.end(); ++y) {

168 {

169 }

170 drawing_area * drawing = new drawing_area(startx, total

171 for (int x = startx ; x < stopx; x++) {

172 color_t c = render_one_pixel (x, y, local_mbox, ser

173 drawing->put_pixel(c);

174 }

175 free(drawing); //Memory Error: use delete instead of f

176 //delete drawing;

Call Stack

find_and_fix_memory_errors.exe!operator()

find_and_fix_memory_errors.exe!execute -

tbb_debug.dll!local_wait_for_all - custom.

tbb_debug.dll!process - arena.cpp:136

tbb_debug.dll!process - market.cpp:181

tbb_debug.dll!run - private_server.cpp:236

tbb_debug.dll!thread_routine - private_ser

tbb_debug.dll!callthreadstartex - threadex

tbb_debug.dll!threadstartex - threadex.c:2

kernel32.dll!BaseThreadInitThunk

Problem State Lifecycle

Makes problems easier to manage

Locate Memory Problems Intel Inspector XE 2013

Target Analysis Type Collection Log Summary

ID	Problem	Object Size	State	Sources	Modules
P1	Mismatched allocation...		New	find_and_fix_memo...	find_a...
P2	Invalid memory access		Not fixed	find_and_fix_memo...	find_a...
P3	Memory not deallocated		Confirmed	api.cpp; util.cpp; vi...	find_a...
P4	Memory leak	1344	Deferred	find_and_fix_memo...	find_a...
P5	Memory leak	784	Fixed	find_and_fix_memo...	find_a...
P6	Memory leak	672	New	find_and_fix_memo...	find_a...

View Source
Edit Source
Copy to Clipboard
Explain Problem
Create Problem Report..
Debug This Problem
Change State
Merge States...

Not fixed
Confirmed
Fixed
Not a problem
Deferred

State	Description
New	Detected by this run
Not Fixed	Previously seen error detected by this run
Not a Problem	Set by user (tool will <u>not</u> change)
Confirmed	Set by user (tool will <u>not</u> change)
Fixed	Set by user (tool <u>will</u> change)
Regression	Error detected with previous state of "Fixed"

Filtering - Focus on what's important

Example: See only the errors in one source file

Before – All Errors

After – Only errors from one source file

Static Analysis Result Intel Inspector XE 2013

Summary

ID	Problem	Sources	State	Weight
P1	Bad free	find_and...	New	80
P2	Divide by zero (poss...	cylinder...	New	75
P7	Unsafe format speci...	parse.cpp	Confirmed	70

Filters: Severity Error 55

Problem: Bad free

Code Locations: Divide by zero (possible)

Description	Source	Function
Divide by zero	cylinder.cpp:131	void cylinder_inter...
129	VCross(&rc, &cyl->axi	
130	VDOT(t, 0, n);	
131	t = - t / ln;	
132	VCross(&n, &cyl->axis	
133	VNorm(&0);	

Source: apigeom.cpp 5

Static Analysis Result Intel Inspector XE 2013

Summary

ID	Problem	Sources	State	Weight
P31	Null pointer derefer...	apigeom...	New	60
P32	Null pointer derefer...	apigeom...	New	60

Filters: Severity Error 5

Problem: Null pointer dereference (po...

Code Locations: Null pointer dereference (possible)

Description	Source	Function	Variable
Memory write	apigeom.cpp...	void rt_sheightfield...	
139	int x, addr;		
140			
141	vertices = (vector *)		
142	normals = (vector *)		
143			

Source: apigeom.cpp 5 item(s)

State: New 5

Suppressed: Not suppressed 5

Investigated: Not investigated 5

(1) Filter – Show only one source file

(2) Error count drops

Tip: Set the "Investigated" filter to "Not investigated" while investigating problems. This removes from view the problems you are done with, leaving only the ones left to investigate.

Static Analysis shown, but filters work the same way for dynamic memory & threading analysis.

Command Line Interface

Automate analysis

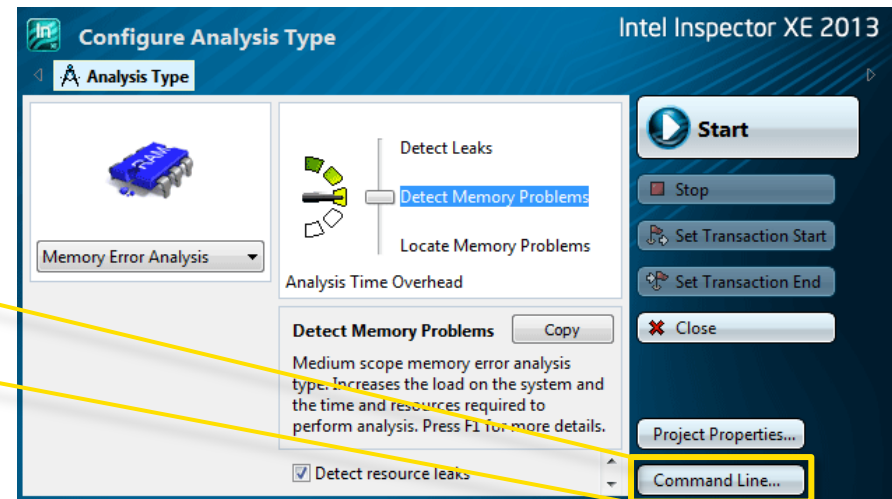
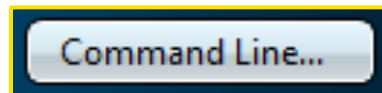
inspxe-cl is the command line:

- **Windows:** C:\Program Files\Intel\Inspector XE \bin[32|64]\inspxe-cl.exe
- **Linux:** /opt/intel/inspector_xe/bin[32|64]/inspxe-cl

Help:

inspxe-cl -help

Set up command line with GUI



Command examples:

1. inspxe-cl -collect-list
2. inspxe-cl -collect ti2 -- MyApp.exe
3. inspxe-cl -report problems

Great for regression analysis – send results file to developer
Command line results can also be opened in the GUI

Intel® Parallel Studio XE Suites

Leading development suite for application performance

	Intel® Cluster Studio XE	Intel® Parallel Studio XE	
Analysis	●	●	Intel® VTune™ Amplifier XE - Performance Profiler
	●	●	Intel® Inspector XE - Memory & Thread Analyzer
	●	●	Static Analysis & Pointer Checker - Find Coding & Security Errors
	●	●	Intel® Advisor XE - Threading Assistant
	●		Intel® Trace Analyzer & Collector - MPI Optimizing Tool
Compilers & Libraries	●	●	Intel® Compiler - Optimizing Compiler for C, C++ and Fortran
	●	●	Intel® Integrated Performance Primitives[†] - Media and Data Optimizations
	●	●	Intel® Threading Building Blocks[†] - Parallelize Applications for Performance
	●	●	Intel® Math Kernel Library - High Performance Math
	●		Intel® MPI Library - Flexible, Efficient and Scalable Messaging

† Available for C, C++ only

C, C++ only and Fortran only versions of Parallel Studio XE are also available.

Create fast, reliable code

Additional Material

Intel® Inspector XE

Product page for [Intel Inspector XE](#) and [Static Analysis](#)

Short demo & “how to” movies:

- [Intel Inspector XE](#) memory and thread checking
- [Static Analysis](#) correctness and security checking
- Cheat sheet on how to set up static analysis: [C, C++](#) and [Fortran](#)

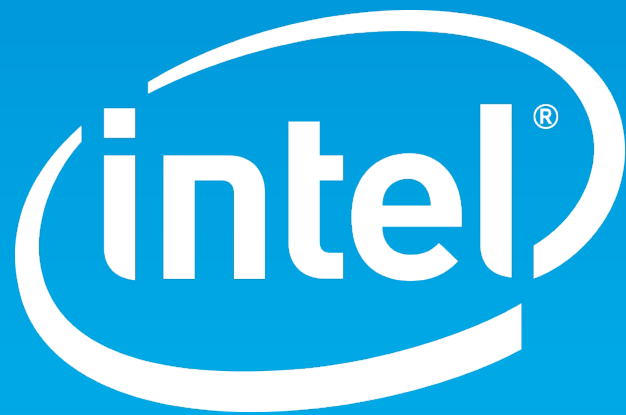
Evaluation Guides – [complete list](#)

- [Eliminate Memory Errors](#)
- [Resolve Resource Leaks](#)
- Static Analysis for [C, C++](#) and [Fortran](#)

Support - [Search Support Articles](#)

More products: [Intel Software Development Products](#)

- [Intel VTune Amplifier XE](#) - performance and thread profiler
- [Intel Advisor XE](#) – threading assistant



Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Backup

Dynamic Analysis Finds Hidden Errors Early

Intel® Inspector XE 2013

Cross-thread Stack Access

Occurs when a thread accesses a different thread's stack.

Data Race

Occurs when multiple threads access the same memory location without proper synchronization and at least one access is a write.

Deadlock

Occurs when two or more threads are waiting for each other to release resources (such as mutexes, critical sections, and thread handles) while holding resources the other threads are trying to acquire. If none of the threads release their resources, then none of the threads can proceed.

GDI Resource Leak

Occurs when a GDI object is created but never deleted.

Incorrect memcpy Call

Occurs when an application calls the memcpy function with two pointers that overlap within the range to be copied. This condition is only checked on Linux* systems. On Windows* systems, this function is safe for overlapping memory.

Invalid Deallocation

Occurs when an application calls a deallocation function with an address that does not correspond to dynamically allocated memory.

Invalid Memory Access

Occurs when a read or write instruction references memory that is logically or physically invalid.

Invalid Partial Memory Access

Occurs when a read or write instruction references a block (2-bytes or more) of memory where part of the block is logically invalid.

Kernel Resource Leak

Occurs when a kernel object handle is created but never closed.

Lock Hierarchy Violation

Occurs when the acquisition order of multiple synchronization objects (such as mutexes, critical sections, and thread handles) in one thread differs from the acquisition order in another thread, and these synchronization objects are owned by the acquiring thread and must be released by the same thread.

Memory Growth

Occurs when a block of memory is allocated but not deallocated within a specific time segment during application execution.

Memory Leak

Occurs when a block of memory is allocated and never released.

Mismatched Allocation/Deallocation

Occurs when a deallocation is attempted with a function that is not the logical reflection of the allocator used.

Missing Allocation

Occurs when an invalid pointer is passed to a deallocation function. The invalid address may point to a previously released heap block.

Thread Start Information

Occurs when the Intel Inspector XE detects the creation of a thread. This *problem* is really informational feedback useful for confirming the number and location of threads created during application execution and data collection.

Unhandled Application Exception

Occurs when the application undergoing analysis crashes because of an unhandled exception thrown by the application.

Uninitialized Memory Access

Occurs when a read of an uninitialized memory location is reported.

Uninitialized Partial Memory Access

Occurs when a read instruction references a block (2-bytes or more) of memory where part of the block is uninitialized.

For details, see our [online documentation](#).

Static Analysis Finds Over 250 Kinds of Errors

Intel® Parallel Studio XE 2013 family of suites

Here are some examples...

- ALLOCATABLE array referenced before allocation
- Argument corresponding to * for width or precision value should be type int

• Argument count mismatch

- Argument count mismatch at call to intrinsic function
- Argument is not a pointer
- Argument type mismatch at call to intrinsic function
- Array parameter element size mismatch

• Array parameter rank mismatch

- Array parameter shape mismatch
- Attempt to violate exception specification
- Bad format flags
- Base class has non-virtual destructor
- Base class lacks destructor
- Big parameter passed by value
- Bounds violation

• Buffer overflow through pointer

- C library routine violates C++ object semantics
- Chunk_size in OpenMP* SCHEDULE clause has side-effects
- Chunk_size in OpenMP* SCHEDULE clause not loop-invariant
- Class has virtual member functions but no derived classes
- COMMON block is partly OpenMP* THREADPRIVATE
- Conditional OpenMP* BARRIER
- Data race
- Data race from cilk_for

• Data race from cilk_spawn

- Destructor contains non-empty exception specification
- Divide by zero
- Double free
- Duplicate subroutine definition
- Exception thrown from destructor
- File closed twice
- Format to argument count mismatch
- Format to argument type mismatch

• FORTRAN IN argument modified

• Function illegally exits OpenMP* construct

- Function result ignored

- Function result not set
- Function return value discarded

• Function use does not match its definition

- Gets function is unsafe
- Global object constructor can throw exception
- Global object destructor can throw exception
- Global redefinition of new or delete
- Global/static variable relies on default initialization
- Illegal parameter value
- Implicit function declaration
- Implicit type conversion causes object slicing

• Improper nesting of OpenMP* constructs

- Improper nesting of OpenMP* CRITICAL directives
- Improper use of intrinsic function
- Improper use of OpenMP* PRIVATE variable
- Improper use of OpenMP* REDUCTION variable
- Improper use of OpenMP* THREADPRIVATE array
- Improper use of OpenMP* THREADPRIVATE variable
- Inconsistent array declaration (element count mismatch)
- Inconsistent array declaration (element size mismatch)
- Inconsistent array declaration (element type mismatch)
- Inconsistent array declaration (size mismatch)
- Inconsistent enumeration declaration (enum value mismatch)
- Inconsistent enumeration declaration (member count mismatch)
- Inconsistent enumeration declaration (name mismatch)
- Inconsistent enumeration declaration (tag mismatch)
- Inconsistent enumeration declaration (type mismatch)

• Inconsistent pointer declaration (size mismatch)

- Inconsistent pointer declaration (target size mismatch)
- Inconsistent pointer declaration (type mismatch)
- Inconsistent string declaration
- Inconsistent structure declaration (field offset mismatch)
- Inconsistent structure/union declaration (field count mismatch)
- Inconsistent structure/union declaration (field name mismatch)
- Inconsistent structure/union declaration (field size mismatch)

• Inconsistent structure/union declaration (field type mismatch)

- Inconsistent structure/union declaration (size mismatch)
- Inconsistent structure/union declaration (tag mismatch)
- Inconsistent structure/union declaration (type mismatch)

For a more complete list, see our [online documentation](#).